Scanning a Keypad with the NEURON® CHIP

LONWORKS™ Engineering Bulletin

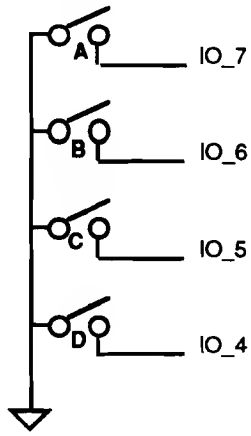
Introduction

This engineering bulletin describes how the Echelon NEURON CHIP can be used to scan a simple 16-key switch matrix to provide a numeric and/or special-function keyboard without the use of a keyboard encoder.

Depending on the number of keys to be scanned and the number of free I/O pins, different solutions are possible.

Scanning up to Eleven Keys

For up to four keys, the simplest way is to connect normally open switches to any of the pins IO_4, IO_5, IO_6, or IO_7 of the NEURON CHIP. These pins have on-chip pull-up resistors that should be enabled by the NEURON C program.



Sample software to drive these switches is shown below. After a switch is pressed, the software waits for some time (10 msec in this example) to allow the switch bounces to subside. The appropriate delay depends on the mechanical construction of the switch and how long it takes to settle in the closed or open position.

```
#pragma enable_io_pullups
IO_4 input bit io_key_A;
IO_5 input bit io_key_B;
IO_6 input bit io_key_C;
IO_7 input bit io_key_D;

when( io_changes( io_key_A ) to 0 ) {     // wait till key is pressed
      delay( 400 );                  // wait 10 milliseconds to debounce
      if( io_in( io_key_A ) == 0 ) {      // key is really pressed
            .......     // handle key press for key A here
      }
}
.....  similarly for keys B, C and D.
```

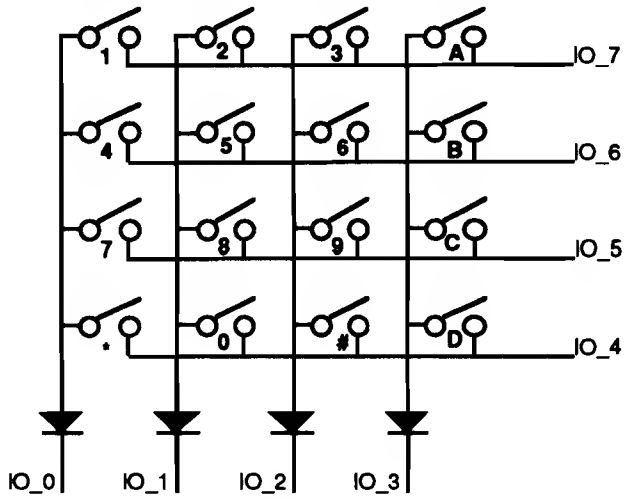
Since each key independently controls a single input pin, this kind of circuit can handle the case where multiple keys are pressed at the same time.

If a keypad has more than four keys (up to 11), the system designer can use the other pins (IO_0 through IO_10) of the NEURON CHIP in the same way. Note that for pins IO_0 through IO_3 and IO_8 through IO_10, there are no on-chip pull-ups. These should be provided with external resistors.

Scanning up to Thirty Keys

But using all the pins in this way may be wasteful if there are other I/O functions that need to be connected to the NEURON CHIP. A more economical way is to connect the keys in a rectangular matrix, using some of the pins to drive the rows of the matrix, and some of the pins to sense the columns. This engineering bulletin describes scanning a 4 X 4 matrix of keys (16 keys). A maximum of 30 keys can be scanned in this way, using all 11 pins to scan the keys in a 6 X 5 matrix.

The NEURON CHIP has 11 general-purpose I/O pins that can be used in several I/O modes. One of these modes is `nibble` mode, where groups of four I/O pins can be read or written together. Below, pins IO_0, IO_1, IO_2 and IO_3 are used as outputs to drive the columns of the switch matrix, and pins IO_4, IO_5, IO_6 and IO_7 as inputs to sense the rows of the switch matrix.



The NEURON CHIP has on-chip pull-up resistors on pins IO_4, IO_5, IO_6 and IO_7, which are enabled by the software statement `#pragma enable_io_pullups`. When none of the keys are pressed, pins IO_4 through IO_7 are therefore pulled up to logic level 1. In the declaration statements, the software configures these four pins as inputs, and pins IO_0, IO_1, IO_2 and IO_3 as outputs. External diodes are connected to these output pins to avoid logic clashes in the event that more than one key in the same row is pressed at the same time.

When none of the keys are pressed, the software writes logic 0's to the four output pins IO_0 through IO_3. When a key is pressed, one of the input pins IO_4 through IO_7 is pulled low, and the NEURON CHIP firmware detects the transition and generates an `io_update_occurs` event, thus activating a task to handle the event. In this example, this task first waits for 10 ms to ensure that any key bounce has subsided. In order to determine which key has been pressed, the software then pulls the output pins IO_0 through IO_3 low one at a time, leaving the other three output pins high, meanwhile reading the input pins each time it does this. If a low input is detected on a pin, this corresponds to the row in which the key is to be found. The row and column number are then used as lookup keys into a two-dimensional array to determine the key code for the key that was pressed.

For example, assume that no key is currently pressed. The output pins are 0 0 0 0, and the input pins are 1 1 1 1. When the 9 key is pressed, the pin IO_5 is pulled low, causing the input to change to 1 1 0 1. The software then scans the keyboard by outputting the following values to the output pins:

Column	Output Written	Input Read		
0	1 1 1 0	1 1 1 1		
1	1 1 0 1	1 1 1 1		
2	1 0 1 1	1 1 0 1	-->	Bit 1 is 0, so key pressed is column 2, row 1

The software updates a network variable `nv_keypad_char` with an ASCII character representing the key that has been pressed. In this example, the characters used are 0 - 9, * and #, corresponding to the layout of a 12-key telephone keypad. The other four keys are coded as A, B, C, and D. When the key is released, the network variable is updated with the ASCII NUL character, which represents the absence of a key press. (The NUL character is different from the '0' character, which is a valid character for this keyboard.) The output network variable is of a standard type - `SNVT_char_ascii` - which allows this node to interoperate with other nodes that accept this type of data from the network.

Code Example for Sixteen Keys

```
//=========================================================================
//
// File:     keypad.nc
// Created: 5/1/91
// Purpose: Scan 16-key keypad
//
//=========================================================================

//-------------------------------------------------------------------------
// Network Variable Objects
//-------------------------------------------------------------------------

network output SNVT_char_ascii nv_keypad_char;

//-------------------------------------------------------------------------
// I/O Objects
//-------------------------------------------------------------------------

#pragma enable_io_pullups

IO_0 output nibble io_keypad_columns = 0;      // pull down all lines
IO_4  input nibble io_keypad_rows;

//-------------------------------------------------------------------------
// Global Variables
//-------------------------------------------------------------------------

const char digit_table[ 4 ][ 4 ] = { "*0#D", "789C", "456B", "123A" };
                                              // map key codes to ASCII
```

```
//-------------------------------------------------------------------------
// IO task
//-------------------------------------------------------------------------

when( io_changes( io_keypad_rows ) ) {
    int row, col, row_val;

    delay( 400 );                                  // 10 msec debounce
    for( col = 0; col < 4; col++ ) {
        io_out( io_keypad_columns, ~( 1 << col ) );
                                                   // pull down one line
        row_val = ~io_in( io_keypad_rows );        // read rows
        for( row = 0; row < 4; row++ ) {           // find which one is zero
            if( row_val & ( 1 << row ) ) {
                nv_keypad_char = digit_table[ row ][ col ];
                                // propagate keypad character to network
                goto done;
            }
        }
    }
    nv_keypad_char = '\0';          // propagate NUL character
done:
    io_out( io_keypad_columns, 0 );          // pull down all lines again
}
```


Echelon Corporation
4015 Miranda Avenue
Palo Alto, CA 94304
Telephone (415) 855-7400
Fax (415) 856-6153

Echelon Europe Ltd
105 Heath Street
London NW3 6SS
England
Telephone (071) 431-1600
Fax (071) 794-0532
International Telephone + 44 71 431-1600
International Fax + 44 71 794-0532

Echelon Japan K.K.
AIOS Gotanda Building #808
10-7, Higashi-Gotanda 1-chome,
Shinagawa-ku, Tokyo 141, Japan
Telephone (03) 3440-8638
Fax (03) 3440-8639

Part Number 005-0004-01